

Münchausen-számok keresése (optimalizálás)

Python3

Kovács Csaba, UBMCGU

Szkriptnyelvek (INBPM9942L) 2021. tavasz

Laborvezető: Dr. Szathmáry László

Áttekintés

- Münchhausen-szám naiv keresés áttekintés
- Egyszerű „gyorsítás”
- Feladat átfogalmazás
- Új algoritmus

Münchausen-számok keresése naivan

- Nagyon lassú akkor ha nagy számig kell vizsgálni (sok az eset) :(

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += 0 if digit == 0 else digit ** digit  
    return accu == num  
  
def list_munchausen(up_to):  
    accu = []  
    for num in range(0, up_to):  
        if is_munchausen(num):  
            accu.append(num)  
    return accu
```

Münchausen-számok keresése naivan

- Nagyon lassú akkor ha nagy számig kell vizsgálni (sok az eset) :(
- Bal oldalt sec-ben a futás idő, középen a felső korlát szám ameddig futott

```
[0.000000s]      1 -> [0]
[0.000000s]     10 -> [0, 1]
[0.000000s]    100 -> [0, 1]
[0.000993s]   1000 -> [0, 1]
[0.020930s]  10000 -> [0, 1, 3435]
[0.225413s] 100000 -> [0, 1, 3435]
[2.535924s] 1000000 -> [0, 1, 3435]
```

```
def is_munchausen(num):
    accu = 0
    for digit_char in str(num):
        digit = int(digit_char)
        accu += 0 if digit == 0 else digit ** digit
    return accu == num

def list_munchausen(up_to):
    accu = []
    for num in range(0, up_to):
        if is_munchausen(num):
            accu.append(num)
    return accu
```

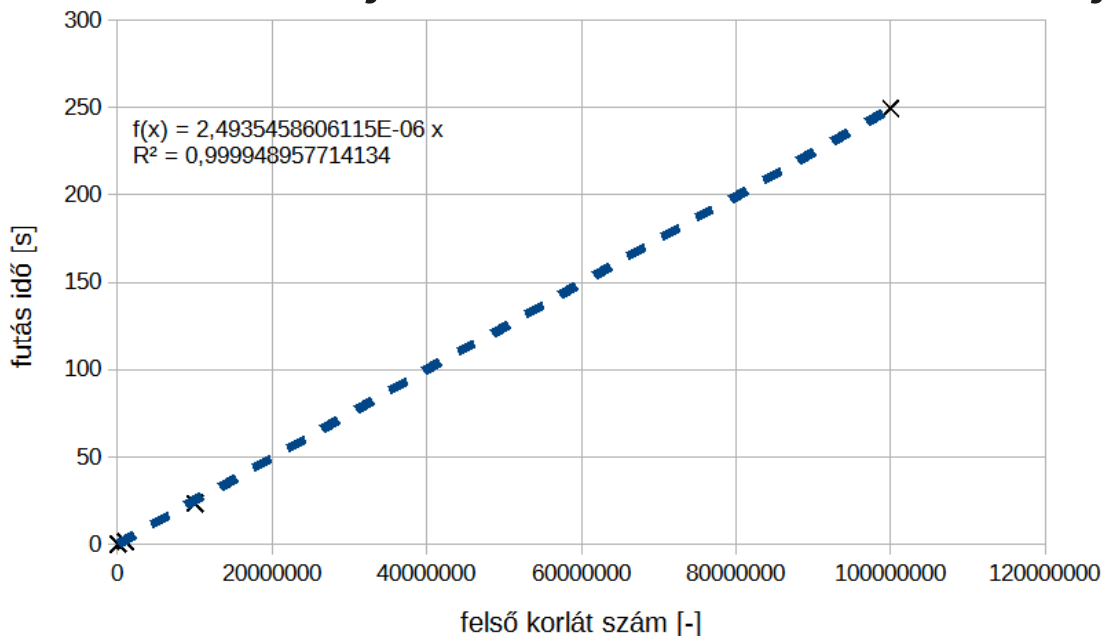
Münchausen-számok keresése naivan

- Ha csak lineáris lenne a felső korlát számra (pl. 10000) már az is para lenne
- De nem is teljesen lineáris hisz a base10 számjegyek számától is függ :(

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += 0 if digit == 0 else digit ** digit  
    return accu == num  
  
def list_munchausen(up_to):  
    accu = []  
    for num in range(0, up_to):  
        if is_munchausen(num):  
            accu.append(num)  
    return accu
```

Münchausen-számok keresése naivan

- Ha csak lineáris lenne a felső korlát számra (pl. 10000) már az is para lenne
- De nem is teljesen lineáris hisz a base10 számjegyek számától is függ :(



```
def is_munchausen(num):
    accu = 0
    for digit_char in str(num):
        digit = int(digit_char)
        accu += 0 if digit == 0 else digit ** digit
    return accu == num

def list_munchausen(up_to):
    accu = []
    for num in range(0, up_to):
        if is_munchausen(num):
            accu.append(num)
    return accu
```

Münchausen-számok keresése naivan

- A feladat 440 000 000-t kért (Nagyon-nagyon sokáig futna)

```
def is_munchausen(num):
    accu = 0
    for digit_char in str(num):
        digit = int(digit_char)
        accu += 0 if digit == 0 else digit ** digit
    return accu == num

def list_munchausen(up_to):
    accu = []
    for num in range(0, up_to):
        if is_munchausen(num):
            accu.append(num)
    return accu
```

Münchausen-számok keresése naivan

- A feladat 440 000 000-t kért (Nagyon-nagyon sokáig futna)
- $440\,000\,000 = 4,4 * 10^8$

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += 0 if digit == 0 else digit ** digit  
    return accu == num  
  
def list_munchausen(up_to):  
    accu = []  
    for num in range(0, up_to):  
        if is_munchausen(num):  
            accu.append(num)  
    return accu
```


Münchausen-számok keresése naivan

- A feladat 440 000 000-t kért (Nagyon-nagyon sokáig futna)
- $440\,000\,000 = 4,4 * 10^8$

k	[s]	[min]
1	0,0	0,0
2	0,0	0,0
3	0,0	0,0
4	0,0	0,0
5	0,2	0,0
6	2,5	0,0
7	24,9	0,4
8	249,4	4,2
9	2493,5	41,6
10	24935,5	415,6
11	249354,6	4155,9

```
def is_munchausen(num):
    accu = 0
    for digit_char in str(num):
        digit = int(digit_char)
        accu += 0 if digit == 0 else digit ** digit
    return accu == num

def list_munchausen(up_to):
    accu = []
    for num in range(0, up_to):
        if is_munchausen(num):
            accu.append(num)
    return accu
```

Münchausen-számok keresése naivan

- Ez túl sok idő.
- Nézzük meg mit is csinálunk, lehet-e valahol spórolni

k	[s]	[min]
1	0,0	0,0
2	0,0	0,0
3	0,0	0,0
4	0,0	0,0
5	0,2	0,0
6	2,5	0,0
7	24,9	0,4
8	249,4	4,2
9	2493,5	41,6
10	24935,5	415,6
11	249354,6	4155,9

```
def is_munchausen(num):
    accu = 0
    for digit_char in str(num):
        digit = int(digit_char)
        accu += 0 if digit == 0 else digit ** digit
    return accu == num

def list_munchausen(up_to):
    accu = []
    for num in range(0, up_to):
        if is_munchausen(num):
            accu.append(num)
    return accu
```

Münchausen-számok keresése naivan

- Végig megyünk a számokon $[0, \text{num}[-$ ig

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += 0 if digit == 0 else digit ** digit  
    return accu == num  
  
def list_munchausen(up_to):  
    accu = []  
    for num in range(0, up_to):  
        if is_munchausen(num):  
            accu.append(num)  
    return accu
```

Münchausen-számok keresése naivan

- Végig megyünk a számokon $[0, \text{num}[-\text{ig}]$
- Minden számot egyenként megvizsgálunk

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += 0 if digit == 0 else digit ** digit  
    return accu == num  
  
def list_munchausen(up_to):  
    accu = []  
    for num in range(0, up_to):  
        if is_munchausen(num):  
            accu.append(num)  
    return accu
```

Münchausen-számok keresése naivan

- Végig megyünk a számokon $[0, \text{num}[$ -ig
- Minden számot egyenként megvizsgálunk
- Minden számjegyet önmaga hatványára emeljük

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += 0 if digit == 0 else digit ** digit  
    return accu == num  
  
def list_munchausen(up_to):  
    accu = []  
    for num in range(0, up_to):  
        if is_munchausen(num):  
            accu.append(num)  
    return accu
```

Münchausen-számok keresése naivan

- Végig megyünk a számokon $[0, \text{num}[$ -ig
- Minden számot egyenként megvizsgálunk
- Minden számjegyet önmaga hatványára emeljük
- Wait a second!

```
def is_munchausen(num):
    accu = 0
    for digit_char in str(num):
        digit = int(digit_char)
        accu += 0 if digit == 0 else digit ** digit
    return accu == num

def list_munchausen(up_to):
    accu = []
    for num in range(0, up_to):
        if is_munchausen(num):
            accu.append(num)
    return accu
```

Münchausen-számok keresése naivan

- Végig megyünk a számokon $[0, \text{num}[-$ ig
- Minden számot egyenként megvizsgáljuk
- Minden számjegyet önmaga hatványára emeljük
- Ez a „hülye gép” kiszámolja egy milliószor a 2^2 -t?

```
def is_munchausen(num):
    accu = 0
    for digit_char in str(num):
        digit = int(digit_char)
        accu += 0 if digit == 0 else digit ** digit
    return accu == num

def list_munchausen(up_to):
    accu = []
    for num in range(0, up_to):
        if is_munchausen(num):
            accu.append(num)
    return accu
```

Münchausen-számok keresése naivan

- Végig megyünk a számokon $[0, \text{num}[-1]$ -ig
- Minden számot egyenként megvizsgáljuk
- Minden számjegyet önmaga hatványára emeljük
- Ez a „hülye gép” kiszámolja egy milliószor a 2^2 -t?
- De miért nem szólt ez a szerencsétlen?
nincs szája :(

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += 0 if digit == 0 else digit ** digit  
    return accu == num  
  
def list_munchausen(up_to):  
    accu = []  
    for num in range(0, up_to):  
        if is_munchausen(num):  
            accu.append(num)  
    return accu
```


Egyszerű gyorsítás

- Ötlet: Mi lenne ha előre be cache-elnénk a hatványokat?

Egyszerű gyorsítás

- Ötlet: Mi lenne ha előre be cache-elnék a hatványokat?
- De mi legyen a cache?

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += CACHE[digit]  
    return accu == num
```

Egyszerű gyorsítás

- Ötlet: Mi lenne ha előre be cache-elnénk a hatványokat?
- De mi legyen a cache?
- Technikailag array-be gyorsan indexelünk... (lista alatt mi van?)

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += CACHE[digit]  
    return accu == num
```

Egyszerű gyorsítás

- Ötlet: Mi lenne ha előre be cache-elnék a hatványokat?
- De mi legyen a cache?
- Technikailag array-be gyorsan indexelünk...

```
def is_munchausen(num):  
    accu = 0  
    for digit_char in str(num):  
        digit = int(digit_char)  
        accu += CACHE[digit]  
    return accu == num
```

- DE MI A CACHE?!?!?!?!?

Egyszerű gyorsítás

- A Cache egy globális változó lesz

```
CACHE =
```

Egyszerű gyorsítás

- Megyünk ugye $[0,10[$ intervallumon

```
CACHE = for x in range(0, 10)
```

Egyszerű gyorsítás

- És önmaga hatványára emelünk mindekit

```
CACHE = x ** x for x in range(0, 10)
```

Egyszerű gyorsítás

- És lista kell

```
CACHE = [x ** x for x in range(0, 10)]
```


Egyszerű gyorsítás

- A magic az, hogy ezt EGYSZER fogjuk elvégezni, és csak beleindexelünk később

```
CACHE = [x ** x for x in range(0, 10)]
```

Egyszerű gyorsítás

- Wait a sec... $0^{**}0 = 1$ de a feladat szerint $0^{**}0 = 0$! BUG!

```
CACHE = [x ** x for x in range(0, 10)]
```

Egyszerű gyorsítás

- Nos, akkor 1-9-ig generáljuk, és rá concat-oljuk egy [0] listára

```
CACHE = [x ** x for x in range(1, 10)]
```

Egyszerű gyorsítás

- Nos, akkor 1-9-ig generáljuk, és rá concat-oljuk egy [0] listára
- Miért is?

```
CACHE = [x ** x for x in range(1, 10)]
```

Egyszerű gyorsítás

- Nos, akkor 1-9-ig generáljuk, és rá concat-oljuk egy [0] listára
- Miért is?

```
CACHE = [x ** x for x in range(1, 10)]
```

```
print([0], [x ** x for x in range(1, 9)]) → [0] [1, 4, 27, 256, 3125, 46656, 823543, 16777216, 387420489]  
print([0] + [x ** x for x in range(1, 9)]) → [0, 1, 4, 27, 256, 3125, 46656, 823543, 16777216, 387420489]
```

Egyszerű gyorsítás

- Nos, akkor 1-9-ig generáljuk, és rá concat-oljuk egy [0] listára
- Akkor tegyünk így

```
CACHE = [0] + [x ** x for x in range(1, 10)]
```

```
print([0], [x ** x for x in range(1, 9)]) → [0] [1, 4, 27, 256, 3125, 46656, 823543, 16777216, 387420489]  
print([0] + [x ** x for x in range(1, 9)]) → [0, 1, 4, 27, 256, 3125, 46656, 823543, 16777216, 387420489]
```

Egyszerű gyorsítás

- Volt értelme a nagy okoskodásunknak, vagy belassítottuk?

Egyszerű gyorsítás

- Volt értelme a nagy okoskodásunknak, vagy belassítottuk?
- Mivel ingadozik a sebesség, ezért lefuttattam mindkettőt százszor `up_to = 100000`-el

Egyszerű gyorsítás

- Volt értelme a nagy okoskodásunknak, vagy belassítottuk?
- Mivel ingadozik a sebesség, ezért lefuttattam mindkettőt százszor `up_to = 100000`-el
- Régi: 23.52 sec
- Új: 10.71 sec

Egyszerű gyorsítás

- Volt értelme a nagy okoskodásunknak, vagy belassítottuk?
- Mivel ingadozik a sebesség, ezért lefuttattam mindkettőt százszor `up_to = 100000`-el
- Régi: 23.52 sec
- Új: 10.71 sec
- Mielőtt elővennénk a pezsgőt...

Egyszerű gyorsítás

- Igazából csak a „konstans szorzót” birizgáljuk, a runtime továbbra is terribilis (és ugye tudjuk hogy valójában közel sem $O(n)$)

k	[s]	[min]
1	0,0	0,0
2	0,0	0,0
3	0,0	0,0
4	0,0	0,0
5	0,2	0,0
6	2,5	0,0
7	24,9	0,4
8	249,4	4,2
9	2493,5	41,6
10	24935,5	415,6
11	249354,6	4155,9

k	[s]	[min]
1	0,0	0,0
2	0,0	0,0
3	0,0	0,0
4	0,0	0,0
5	0,1	0,0
6	1,2	0,0
7	12,5	0,2
8	124,7	2,1
9	1246,8	20,8
10	12467,7	207,8
11	124677,3	2078,0

Feladat átfogalmazás

- Nézzük meg újból

Feladat átfogalmazás

- Nézzük meg újból
- Egy természetes számot akkor nevezünk Münchausen-számnak, ha minden egyes számjegyét az adott számjegy által meghatározott hatványra emelve, majd ezen hatványok összegét véve az eredeti számot kapjuk vissza.
- Most legyen $0^0 = 0$

Feladat átfogalmazás

- Érdekes
- De van-e más is ami „kiderül”?

Feladat átfogalmazás

- Hmh... nézzük már meg 2021-et!

2 0 2 1

Feladat átfogalmazás

- Hmh... nézzük már meg 2021-et!

$$\begin{array}{cccc} 2 & 0 & 2 & 1 \\ 2^2 & 0^0 & 2^2 & 1^1 \end{array}$$

Feladat átfogalmazás

- Hmh... nézzük már meg 2021-et!

$$\begin{array}{cccc} 2 & 0 & 2 & 1 \\ 2^2 & 0^0 & 2^2 & 1^1 \\ 4+ & 0+ & 4+ & 1 \end{array}$$

Feladat átfogalmazás

- Hmh... nézzük már meg 2021-et!

$$\begin{array}{cccc} 2 & 0 & 2 & 1 \\ 2^2 & 0^0 & 2^2 & 1^1 \\ 4+ & 0+ & 4+ & 1 = 9 \end{array}$$

Feladat átfogalmazás

- $2021 \Rightarrow 9$

$$\begin{array}{cccc} 2 & 0 & 2 & 1 \\ 2^2 & 0^0 & 2^2 & 1^1 \\ 4+ & 0+ & 4+ & 1 = 9 \end{array}$$

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- Nézzük meg 1022-t

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- Nézzük meg 1022-t

1 0 2 2

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- Nézzük meg 1022 -t

$$\begin{array}{cccc} 1 & 0 & 2 & 2 \\ 1^1 & 0^0 & 2^2 & 2^2 \end{array}$$

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- Nézzük meg 1022 -t

1 0 2 2

1^1 0^0 2^2 2^2

1+ 0+ 4+ 4

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- Nézzük meg 1022 -t

1 0 2 2

1^1 0^0 2^2 2^2

$1 + 0 + 4 + 4 = 9$

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$
- Hmmhh...

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$
- Hmmhh...
- A hatvány szummájuk ugyanaz!

Feladat átfogalmazás

- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$
- Hmmhh...
- A hatvány szummájuk ugyanaz!
- Hatvány szummában csak a számjegyek darabszáma számít!


Feladat átfogalmazás

- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$
- Hmmhh...
- A hatvány szummájuk ugyanaz!
- Hatvány szummában csak a számjegyek darabszáma számít!

$$2^2 \ 0^0 \ 2^2 \ 1^1 = 9$$

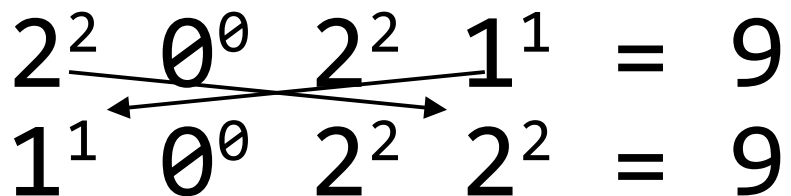
Feladat átfogalmazás

- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$
- Hmmhh...
- A hatvány szummájuk ugyanaz!
- Hatvány szummában csak a számjegyek darabszáma számít!

$$2^2 \quad 0^0 \quad 2^2 \quad 1^1 = 9$$


Feladat átfogalmazás


- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$
- Hmmhh...
- A hatvány szummájuk ugyanaz!
- Hatvány szummában csak a számjegyek darabszáma számít!

$$\begin{array}{ccccccc} 2^2 & 0^0 & 2^2 & 1^1 & = & 9 \\ 1^1 & 0^0 & 2^2 & 2^2 & = & 9 \end{array}$$


Feladat átfogalmazás

- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$
- Hmmhh...
- A hatvány szummájuk ugyanaz!
- Hatvány szummában csak a számjegyek darabszáma számít!

$$2^2 \quad 0^0 \quad 2^2 \quad 1^1 = 9$$

$$1^1 \quad 0^0 \quad 2^2 \quad 2^2 = 9$$


Feladat átfogalmazás

- $2021 \Rightarrow 9$
- $1022 \Rightarrow 9$
- Hmmhh...
- A hatvány szummájuk ugyanaz!
- Hatvány szummában csak a számjegyek darabszáma számít!

$$2^2 \quad 0^0 \quad 2^2 \quad 1^1 = 9$$

$$1^1 \quad 0^0 \quad 2^2 \quad 2^2 = 9$$

$$1^1 \quad 2^2 \quad 0^0 \quad 2^2 = 9$$


Feladat átfogalmazás

- Hány olyan (négyjegyű) szám van amiben 1 db 0, 1 db 1, 2 db 2 van?

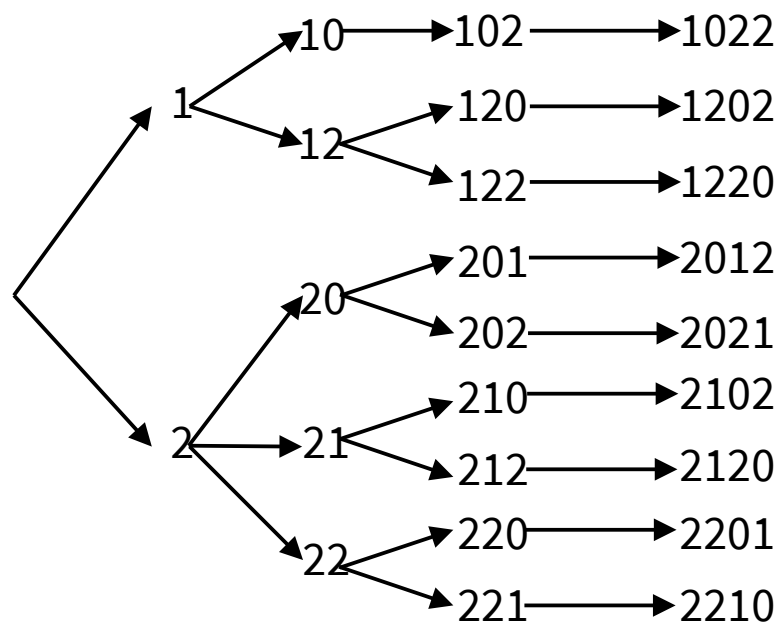
Feladat átfogalmazás

- Hány olyan (négyjegyű) szám van amiben 1 db 0, 1 db 1, 2 db 2 van?

$$\frac{3!}{1!*1!*1!} + \frac{3!}{1!*1!*2!} = 3 + 6 = 9$$

Feladat átfogalmazás

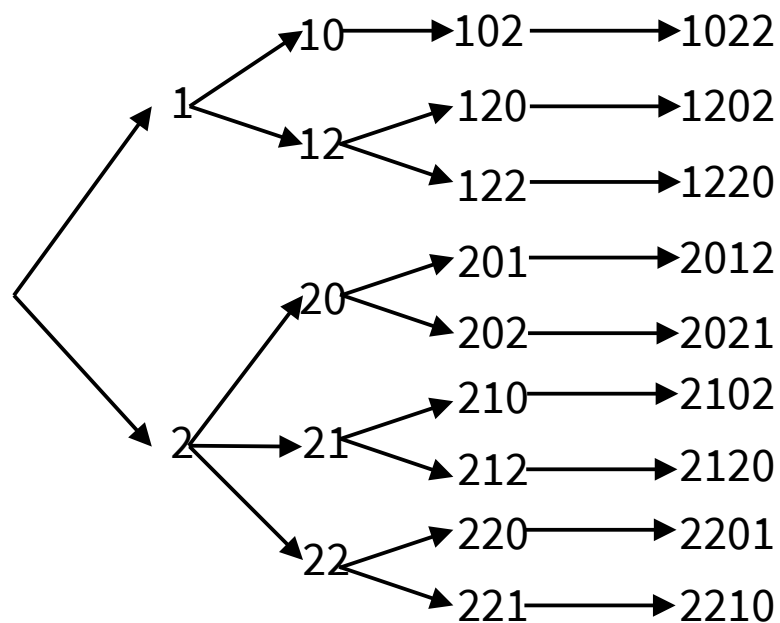
- Hány olyan (négyjegyű) szám van amiben 1 db 0, 1 db 1, 2 db 2 van?



$$\frac{3!}{1!*1!*1!} + \frac{3!}{1!*1!*2!} = 3 + 6 = 9$$

Feladat átfogalmazás

- Hány olyan (négyjegyű) szám van amiben 1 db 0, 1 db 1, 2 db 2 van?



$$\frac{3!}{1! \cdot 1! \cdot 1!} + \frac{3!}{1! \cdot 1! \cdot 2!} = 3 + 6 = 9$$

2021 1022

2012 1022

2201 1202

2102 1202

2120 1220

2210 1220

Sorrendje nem számít a 2-eseknek

Feladat átfogalmazás

- Oh ok, de mit is nyerünk ezzel?

1022

1202

1220

2012

2021

2102

2120

2201

2210

Feladat átfogalmazás

- Oh ok, de mit is nyerünk ezzel?

1022 -> 9

1202 -> 9

1220 -> 9

2012 -> 9

2021 -> 9

2102 -> 9

2120 -> 9

2201 -> 9

2210 -> 9

Feladat átfogalmazás

- Oh ok, de mit is nyerünk ezzel?

1022 -> 9

1202 -> 9

1220 -> 9

2012 -> 9

2021 -> 9

2102 -> 9

2120 -> 9

2201 -> 9

2210 -> 9

A hatvány szumma
szempontjából ugyanazok...

Feladat átfogalmazás

- Valahogy „egyben” kéne kiszámítani ezeket a „csoportokat”...

1022 -> 9

1202 -> 9

1220 -> 9

2012 -> 9

2021 -> 9

2102 -> 9

2120 -> 9

2201 -> 9

2210 -> 9

A hatvány szumma
szempontjából ugyanazok...

Feladat átfogalmazás

- Mi lenne ha azt mondanám:

Feladat átfogalmazás

- Mi lenne ha azt mondanám:
- A számokon túl létezik egy másik világ is



Feladat átfogalmazás

- Számok helyett, listákat fogunk vizsgálni
- A listában számjegyek előfordulási száma lesz
- 0-tól 9-ig

[„0” count, „1” count, „2” count, ... „8” count, „9” count]

Feladat átfogalmazás

- Nézzük meg mit reprezentál majd a lista

Feladat átfogalmazás

- Példa: 2021

0 1 2 3 4 5 6 7 8 9
[]

Feladat átfogalmazás

- Példa: 2021 1 darab 0

0 1 2 3 4 5 6 7 8 9
[1]

Feladat átfogalmazás

- Példa: 2021 1 darab 1

0	1	2	3	4	5	6	7	8	9
[1	1]

Feladat átfogalmazás

- Példa: 2021 2 darab 2

0	1	2	3	4	5	6	7	8	9
[1	1	2	0	0	0	0	0	0]

Feladat átfogalmazás

- De persze közben ezt is: 2201

0	1	2	3	4	5	6	7	8	9
[1	1	2	0	0	0	0	0	0]

Feladat átfogalmazás

- De persze közben ezt is: **2201**

0	1	2	3	4	5	6	7	8	9
[1	1	2	0	0	0	0	0	0	0]

- Vagy ezt is: **1220**

0	1	2	3	4	5	6	7	8	9
[1	1	2	0	0	0	0	0	0	0]

Feladat átfogalmazás

- De persze közben ezt is: **2201**

0	1	2	3	4	5	6	7	8	9
[1	1	2	0	0	0	0	0	0]

- Vagy ezt is: **1220**

0	1	2	3	4	5	6	7	8	9
[1	1	2	0	0	0	0	0	0]

- Meg ezt is: **2120**

0	1	2	3	4	5	6	7	8	9
[1	1	2	0	0	0	0	0	0]



Feladat átfogalmazás

- Az algoritmus számok helyett ilyen 10 elemű vektorokon fog iterálni

Feladat átfogalmazás

- Az algoritmus számok helyett ilyen 10 elemű vektorokon fog iterálni
- Nem egy szám lesz a határ (pl.: 440 000 000), hanem az hogy maximum hány számjegyű számokat vizsgálunk (legyen „places”)

Feladat átfogalmazás

- Az algoritmus számok helyett ilyen 10 elemű vektorokon fog iterálni
- Nem egy szám lesz a határ (pl.: 440 000 000), hanem az hogy maximum hány számjegyű számokat vizsgálunk (legyen „places”)
- Na jó, de így mi lesz a határ? (És ugye egy vektorra/listára kell megfogalmazni)

Feladat átfogalmazás

- Az algoritmus számok helyett ilyen 10 elemű vektorokon fog iterálni
- Nem egy szám lesz a határ (pl.: 440 000 000), hanem az hogy maximum hány számjegyű számokat vizsgálunk (legyen „places”)
- Na jó, de így mi lesz a határ? (És ugye egy vektorra/listára kell megfogalmazni)
- Pl.: Max 9 számjegyű számokat vizsgálunk → vektor sum-ja max 9 lehet

Például

```
0 1 2 3 4 5 6 7 8 9  
[0 9 0 0 0 0 0 0 0 0]
```

Vagy mondjuk

```
0 1 2 3 4 5 6 7 8 9  
[0 8 1 0 0 0 0 0 0 0]
```


Feladat átfogalmazás

- DE MIRE JÓ???????

Feladat átfogalmazás

- DE MIRE JÓ???????
- Nos, hány számot kéne megvizsgálni a régi módon ha azt mondják:
Nézd meg max 5 számjegyre?
0-99999 azaz 100 000 darab

Feladat átfogalmazás

- DE MIRE JÓ???????
- Nos, hány számot kéne megvizsgálni a régi módon ha azt mondják:
Nézd meg max 5 számjegyre?
0-99999 azaz 100 000 darab
- Új módon?

Feladat átfogalmazás

- DE MIRE JÓ???????
- Nos, hány számot kéne megvizsgálni a régi módon ha azt mondják:
Nézd meg max 5 számjegyre?
0-99999 azaz 100 000 darab
- Új módon?

$$\left(\binom{10+1-1}{1}-1\right)+\left(\binom{10+2-1}{2}-1\right)+\left(\binom{10+3-1}{3}-1\right)+\left(\binom{10+4-1}{4}-1\right)+\left(\binom{10+5-1}{5}-1\right)+1=2998$$

Feladat átfogalmazás

- DE MIRE JÓ???????
- Nos, hány számot kéne megvizsgálni a régi módon ha azt mondják:
Nézd meg max 5 számjegyre?
0-99999 azaz 100 000 darab

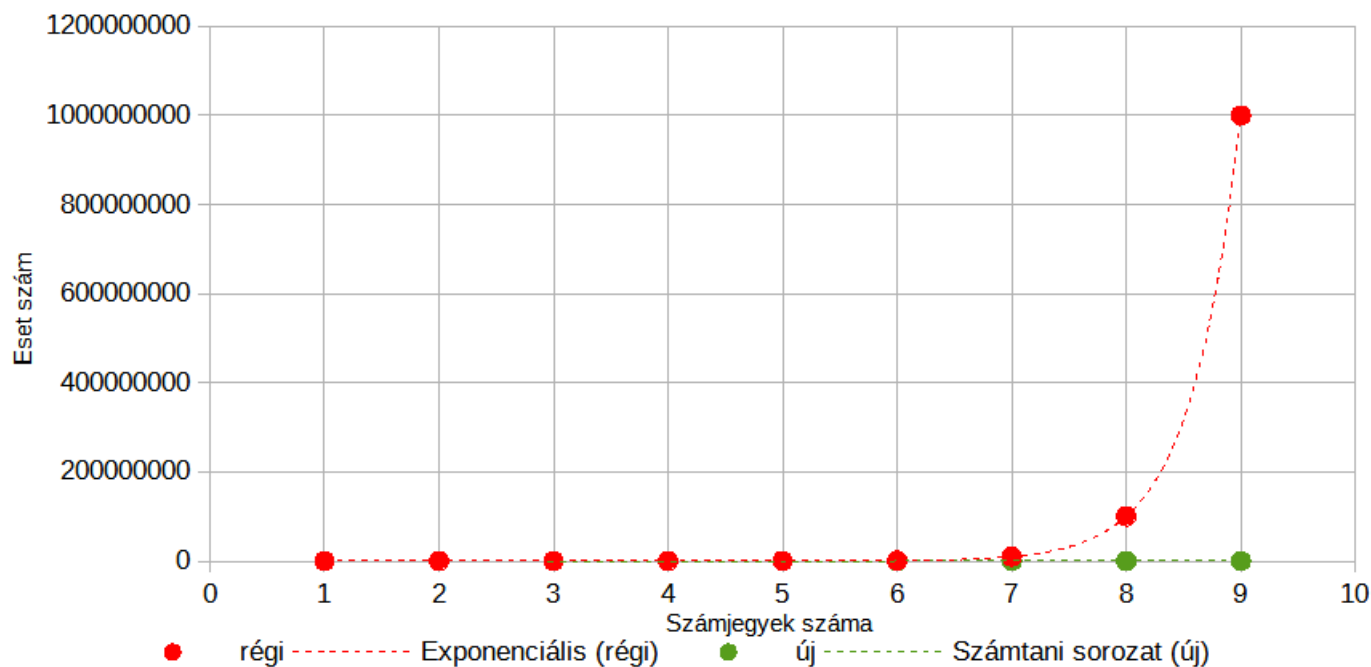
- Új módon?

$$\left(\binom{10+1-1}{1}-1\right)+\left(\binom{10+2-1}{2}-1\right)+\left(\binom{10+3-1}{3}-1\right)+\left(\binom{10+4-1}{4}-1\right)+\left(\binom{10+5-1}{5}-1\right)+1=2998$$

- Egyszerűbb ábrával belátni hogy jóval kevesebb lesz

Feladat átfogalmazás

- A régihez képest szignifikánsan kevesebb esetet kell vizsgálni



Feladat átfogalmazás

- Nézzük meg, hogy ha vektorokat használunk akkor ki tudjuk-e hozni a Münchausen-számokat!

Feladat átfogalmazás

- Most képzeljük el, hogy kitöltött egy vektort...

Feladat átfogalmazás

- Most képzeljük el, hogy kitöltött egy vektort...
- Mondjuk

0	1	2	3	4	5	6	7	8	9
[0	0	0	2	1	1	0	0	0	0]

Feladat átfogalmazás

- Most képzeljük el, hogy kitöltött egy vektort...

- Mondjuk

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ebből simán tudjuk számolni a szummát (sőt, akár közben is :)

- $2 * 3^3$

Feladat átfogalmazás

- Most képzeljük el, hogy kitöltött egy vektort...

- Mondjuk

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ebből simán tudjuk számolni a szummát (sőt, akár közben is :)

- $2 * 3^3 + 1 * 4^4$

Feladat átfogalmazás

- Most képzeljük el, hogy kitöltött egy vektort...

- Mondjuk

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ebből simán tudjuk számolni a szummát (sőt, akár közben is :)

- $2 * 3^3 + 1 * 4^4 + 1 * 5^5$

Feladat átfogalmazás

- Most képzeljük el, hogy kitöltött egy vektort...

- Mondjuk

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ebből simán tudjuk számolni a szummát (sőt, akár közben is :)
- $2 * 3^3 + 1 * 4^4 + 1 * 5^5$
- Ok, meg van a hatvány szumma, hasonlítsuk akkor össze a számmal!

Feladat átfogalmazás

- Most képzeljük el, hogy kitöltött egy vektort...

- Mondjuk

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ebből simán tudjuk számolni a szummát (sőt, akár közben is :)
- $2 * 3^3 + 1 * 4^4 + 1 * 5^5$
- Ok, meg van a hatvány szumma, hasonlítsuk akkor össze a számmal!
- DE MOST MÁR MEG NINCS SZÁM AMIVEL ÖSSZEHASONLÍTSUK!

Feladat átfogalmazás

- Most képzeljük el, hogy kitöltött egy vektort

- Mondjuk

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ebből simán tudjuk számolni a szummát
- $2 * 3^3 + 1 * 4^4 + 1 * 5^5$
- Ok, meg van a hatvány szumma, hasonlítsuk akkor össze a számmal!
- DE MOST MÁR MEG NINCS SZÁM AMIVEL ÖSSZEHASONLÍTSUK!
- RIP RUN?!

Feladat átfogalmazás

- Mi lenne ha azt mondanám:

Feladat átfogalmazás

- Mi lenne ha azt mondanám:
- Ha valóban ő a kiválasztott akkor azt tudni fogja önmagáról ?



Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0	1	2	3	4	5	6	7	8	9
[0	0	0	2	1	1	0	0	0	0]

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma

$$2 * 3^3 + 1 * 4^4 + 1 * 5^5$$

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM

$$2 * 3^3 + 1 * 4^4 + 1 * 5^5 = 3435$$

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM
- Dehát akkor SZÁM és hatvány_szumma „ugyanolyan számjegyekből áll”
(hisz egyenlőek)

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM
- Dehát akkor SZÁM és hatvány_szumma „ugyanolyan számjegyekből áll” (hisz egyenlőek)
- Azaz ha a hatvány szummát kiszámoljuk

[0 0 0 2 1 1 0 0 0 0] => 3435

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM
- Dehát akkor SZÁM és hatvány_szumma „ugyanolyan számjegyekből áll”
(hisz egyenlőek)
- Azaz ha a hatvány szummát kiszámoljuk
- Majd leszámoljuk a számjegyeit a vektorból

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0] => 3435

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM
- Dehát akkor SZÁM és hatvány_szumma „ugyanolyan számjegyekből áll”
(hisz egyenlőek)
- Azaz ha a hatvány szummát kiszámoljuk
- Majd leszámoljuk a számjegyeit a vektorból

0 1 2 3 4 5 6 7 8 9
[0 0 0 1 1 1 0 0 0 0] => 3435

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM
- Dehát akkor SZÁM és hatvány_szumma „ugyanolyan számjegyekből áll”
(hisz egyenlőek)
- Azaz ha a hatvány szummát kiszámoljuk
- Majd leszámoljuk a számjegyeit a vektorból

0 1 2 3 4 5 6 7 8 9
[0 0 0 1 0 1 0 0 0 0] => 3435

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM
- Dehát akkor SZÁM és hatvány_szumma „ugyanolyan számjegyekből áll” (hisz egyenlőek)
- Azaz ha a hatvány szummát kiszámoljuk
- Majd leszámoljuk a számjegyeit a vektorból

0 1 2 3 4 5 6 7 8 9
[0 0 0 0 0 1 0 0 0 0] => 3435

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM
- Dehát akkor SZÁM és hatvány_szumma „ugyanolyan számjegyekből áll”
(hisz egyenlőek)
- Azaz ha a hatvány szummát kiszámoljuk
- Majd leszámoljuk a számjegyeit a vektorból

0 1 2 3 4 5 6 7 8 9
[0 0 0 0 0 0 0 0 0 0] => 3435

Feladat átfogalmazás

- Mármint tegyük fel ez a vektor

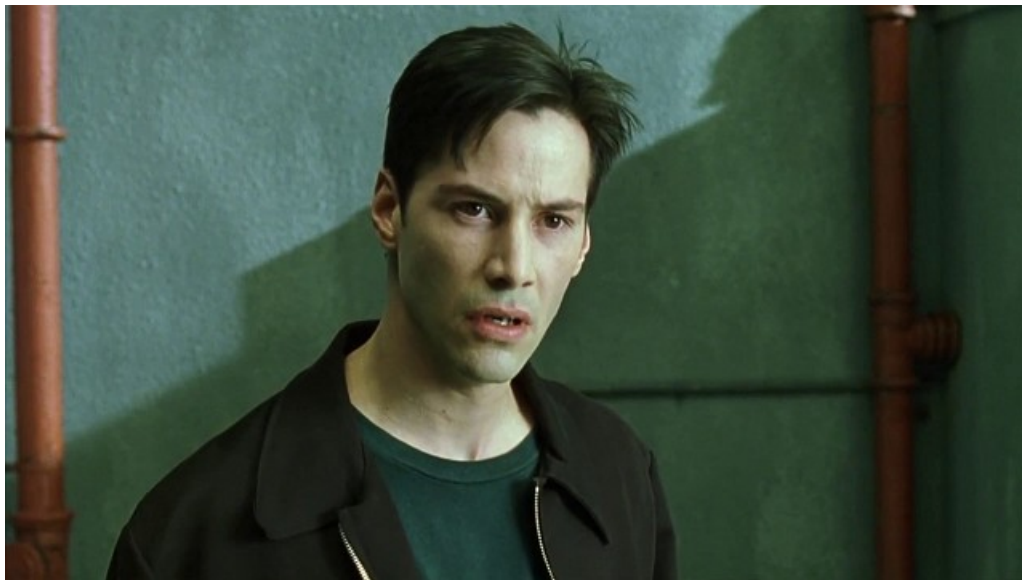
0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0]

- Ha ez Münchausen-szám
- Akkor hatvány_szumma MAGA A SZÁM
- Dehát akkor SZÁM és hatvány_szumma „ugyanolyan számjegyekből áll” (hisz egyenlőek)
- Azaz ha a hatvány szummát kiszámoljuk
- Ha kinullázza a vektort akkor

0 1 2 3 4 5 6 7 8 9
[0 0 0 0 0 0 0 0 0 0] => 3435

Feladat átfogalmazás

- A hatvány szumma maga a szám 3435



Feladat átfogalmazás

- A hatvány szumma maga a szám
- Szóval a $\text{sum} == \text{szám}$ helyett az lesz a tesztünk hogy a hatvány szumma számjegy számát leszámoljuk az öt létrehozó vektorból
- Ha a vektor összes eleme 0 \rightarrow Münchausen-szám
- Ha csak egy eleme is nem 0 \rightarrow Nem Münchausen-szám

Feladat átfogalmazás

- Generáljunk 10 elemű vektorokat (közben adogassuk össze a hatványokat)!

Feladat átfogalmazás

- Generáljunk 10 elemű vektorokat (közben adogassuk össze a hatványokat)!
- Mikor teli a vektor (és kész a szumma), akkor teszteljük őket!

Megoldás terv

- Generáljunk 10 elemű vektorokat (közben adogassuk össze a hatványokat)!
- Mikor teli a vektor (és kész a szumma), akkor teszteljük őket!
- Ha a szummában ugyanannyi számjegy van mint a vektorban akkor megoldás!

is_munchausen Teszt

- Nézzük a tesztelő funkciót!

is_munchausen Teszt

- Vegyünk egy listát „digit_counts” és a hatvány szummát „pow_sum”

```
def is_munchausen(digit_counts, pow_sum):
```

is_munchausen Teszt

- Iteráljunk a számjegyek karakterein!

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):
```

is_munchausen Teszt

- Váltssuk át a számjegy karaktert számmá!

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)
```

is_munchausen Teszt

- Vonjunk ki egyet a lista digit helyéről!

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Vonjunk ki egyet a lista digit helyéről!
- Miért is?

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```


is_munchausen Teszt

- Vonjunk ki egyet a lista digit helyéről!

- Miért is?

0 1 2 3 4 5 6 7 8 9
[0 0 0 2 1 1 0 0 0 0] => 3435

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Vonjunk ki egyet a lista digit helyéről!

- Miért is?

0 1 2 3 4 5 6 7 8 9
[0 0 0 1 1 1 0 0 0 0] => 3435

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Vonjunk ki egyet a lista digit helyéről!

- Miért is?

0 1 2 3 4 5 6 7 8 9
[0 0 0 1 0 1 0 0 0 0] => 3435

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Vonjunk ki egyet a lista digit helyéről!

- Miért is?

0 1 2 3 4 5 6 7 8 9
[0 0 0 0 0 1 0 0 0 0] => 3435

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Vonjunk ki egyet a lista digit helyéről!

- Miért is?

0 1 2 3 4 5 6 7 8 9
[0 0 0 0 0 0 0 0 0 0] => 3435

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Az maradt hátra, hogy megnézzük végig nulla-e a lista

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Az maradt hátra, hogy megnézzük végig nulla-e a lista

- Mármint pl.:
0 1 2 3 4 5 6 7 8 9
[0 1 3 0 0 0 0 0 0 0] =>

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Az maradt hátra, hogy megnézzük végig nulla-e a lista
- Mármint pl.:
0 1 2 3 4 5 6 7 8 9
[0 1 3 0 0 0 0 0 0 0] => 13 (mivel $1 * 1^1 + 3 * 2^2$)

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```


is_munchausen Teszt

- Az maradt hátra, hogy megnézzük végig nulla-e a lista

- Mármint pl.:
0 1 2 3 4 5 6 7 8 9
[0 0 3 0 0 0 0 0 0 0] => 13

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Az maradt hátra, hogy megnézzük végig nulla-e a lista
- Mármint pl.:
0 1 2 3 4 5 6 7 8 9
[0 0 3 -1 0 0 0 0 0 0] => 13

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Az maradt hátra, hogy megnézzük végig nulla-e a lista
- Mármint pl.:
0 1 2 3 4 5 6 7 8 9
[0 0 3 -1 0 0 0 0 0 0] => 13 nem nullázta ki a listát

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1
```

is_munchausen Teszt

- Az maradt hátra, hogy megnézzük végig nulla-e a lista
- Csak végig megyünk és ha valaki nem 0 False, egyébként True)

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1  
    for digit_count in digit_counts:  
        if digit_count != 0:  
            return False  
    return True
```

is_munchausen Teszt

- Nagyon ok...teszteljük

(link: https://github.com/rkeeves/munchausen/blob/main/src/e07_munchausen_property.py)

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1  
    for digit_count in digit_counts:  
        if digit_count != 0:  
            return False  
    return True
```

is_munchausen Teszt

- Nagyon ok...teszteljük
- Ok-nak néz ki

```
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0] -> 0 -> True  
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0] -> 1 -> True  
[0, 0, 0, 2, 1, 1, 0, 0, 0, 0] -> 3435 -> True  
[1, 1, 2, 0, 0, 0, 0, 0, 0, 0] -> 9 -> False
```

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1  
    for digit_count in digit_counts:  
        if digit_count != 0:  
            return False  
    return True
```

is_munchausen Teszt

- Nagyon ok...teszteljük
- Ok-nak néz ki
- Jöjjön a hatvány szumma számítás!

```
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0] -> 0 -> True  
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0] -> 1 -> True  
[0, 0, 0, 2, 1, 1, 0, 0, 0, 0] -> 3435 -> True  
[1, 1, 2, 0, 0, 0, 0, 0, 0, 0] -> 9 -> False
```

```
def is_munchausen(digit_counts, pow_sum):  
    for digit_ch in str(pow_sum):  
        digit = int(digit_ch)  
        digit_counts[digit] -= 1  
    for digit_count in digit_counts:  
        if digit_count != 0:  
            return False  
    return True
```

pow_sum

- Csináljunk egy kis segéd függvényt ami a kész listából kiszámítja a hatvány szummát!

pow_sum

- Vegyünk egy listát!

```
def compute_pow_sum(digit_counts):  
    return
```

pow_sum

- Ugye szummázni fogunk....

```
def compute_pow_sum(digit_counts):  
    return sum(  
        |  
    )
```

pow_sum

- A darabszámokon fogunk végigiterálni az biztos...

```
def compute_pow_sum(digit_counts):  
    return sum(  
        for digit_count in digit_counts  
    )
```

pow_sum

- És ugye meg kell darab számmal szorozni az „önmaga hatvány”-t
- Pl: $2 * 3^3 + 1 * 4^4 + 1 * 5^5$

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count *          for digit_count in          digit_counts  
    )
```

pow_sum

- De hol a digit amit hatványozunk? :((((((((

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count *          for digit_count in digit_counts  
    )
```

pow_sum

- Mi legyen a digit?

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count *          for digit_count in          digit_counts  
    )
```

pow_sum

- Mi legyen a digit?
- A sorszám a listában!

```
0 1 2 3 4 5 6 7 8 9  
[0 0 3 0 0 0 0 0 0 0]
```

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count *          for digit_count in          digit_counts  
    )
```

pow_sum

- Mi legyen a digit?
- A sorszám a listában!
- Tanultunk-e olyan dolgot ami az indexet is kiadja?

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count *          for          digit_count in          digit_counts  
    )
```


pow_sum

- De mi legyen a digit?
- A sorszáma a listában!
- Tanultunk-e olyan dolgot ami az indexet is kiadja?
- enumerate

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count *          for digit, digit_count in enumerate(digit_counts)  
    )
```

pow_sum

- Ugye az önmaga hatványra emlékszünk hogy CACHE-eltük
- Azaz ez ekvivalens `digit ** digit` -el

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count * CACHE[digit] for digit, digit_count in enumerate(digit_counts)  
    )
```

pow_sum

- Kész... (már teszteltük az előzőekben „under the hood”, csak nem szóltam :)

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count * CACHE[digit] for digit, digit_count in enumerate(digit_counts)  
    )
```

pow_sum

- Kész...

(link: https://github.com/rkeeves/munchausen/blob/main/src/e07_munchausen_property.py)

```
def compute_pow_sum(digit_counts):  
    return sum(  
        digit_count * CACHE[digit] for digit, digit_count in enumerate(digit_counts)  
    )
```

Generálás

- Hogyan lehetne generálni szépen „sorban” ilyen vektorokat?
- Mármint user mondjuk 10 számjegyre kérné...
- Bontsuk ketté a problémát
- Először írjunk egy függvényt ami adott számjegyre számokat vizsgál mondjuk az összes n számjegyre (*list_munchausens_of_decimal_places*)
- Majd írjunk egy másikat, ami ha mondjuk 3-at kap, akkor meghívja sorban
list_munchausens_of_decimal_places(1)
list_munchausens_of_decimal_places(2)
list_munchausens_of_decimal_places(3)

Inner functions

- Apró kitérő kell, elnézést (ez meg fogja törni a lendületet...ha még maradt :)
- Mivel még nem tanultuk, a belső függvényekről szót kell ejteni
- *(De nem adjuk vissza hívónak a function-t, szóval closure nem annyira kell)*

Inner functions

- Írjunk már egy kicsi progit faktoriális számításra rekurzióval!

Inner functions

- Írjunk már egy kicsi progit faktoriális számításra rekurzióval!
- Plot Twist: Ha $n! \geq 5000$, akkor adjunk vissza 5000-et!

Inner functions

- Írjunk már egy kicsi progit faktoriális számításra rekurzióval!
- Plot Twist: Ha $n! \geq 5000$, akkor adjunk vissza 5000-et!
- Plusz a függvény bármit is csinál az elején printelje már ki a saját nevét és hogy mivel hívták (később jól fog jönni)

Inner functions

- Vegyük egy számot, ehhez keressük a num! faktoriálist

```
def fact_ver1(num):
```

Inner functions

- Csak kiprinteljük magunkat

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))
```

Inner functions

- $0! == 1$ és $1! == 1$, ilyenkor az eredmény easy :) (base cases are the BEST cases)

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1
```

Inner functions

- Egyébként meg ugye $n! = n * (n-1)!$

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1
```

Inner functions

- Egyébként meg ugye $n! = n * (n-1)!$

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1  
    ret = num * fact_ver1(num - 1)
```

Inner functions

- DE: Ha $n! \geq 5000$, akkor adjunk vissza 5000-et!

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1  
    ret = num * fact_ver1(num - 1)  
    if ret >= 5000:  
        return 5000  
    return ret
```

Inner functions

- Ok, nézzük meg....

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1  
    ret = num * fact_ver1(num - 1)  
    if ret >= 5000:  
        return 5000  
    return ret
```


Inner functions

- Teszteljük 8-ra

(link: https://github.com/rkeeves/munchausen/blob/main/src/fact_ver1.py)

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1  
    ret = num * fact_ver1(num - 1)  
    if ret >= 5000:  
        return 5000  
    return ret
```

Inner functions

- Teszteljük 8-ra

```
fact_ver1(8)
fact_ver1(7)
fact_ver1(6)
fact_ver1(5)
fact_ver1(4)
fact_ver1(3)
fact_ver1(2)
fact_ver1(1)
```

```
def fact_ver1(num):
    print("fact_ver1({})".format(num))
    if num < 2:
        return 1
    ret = num * fact_ver1(num - 1)
    if ret >= 5000:
        return 5000
    return ret
```

Inner functions

- Teszteljük 8-ra
- Hát ez végig hívja magát 1-ig

```
fact_ver1(8)
fact_ver1(7)
fact_ver1(6)
fact_ver1(5)
fact_ver1(4)
fact_ver1(3)
fact_ver1(2)
fact_ver1(1)
```

```
def fact_ver1(num):
    print("fact_ver1({})".format(num))
    if num < 2:
        return 1
    ret = num * fact_ver1(num - 1)
    if ret >= 5000:
        return 5000
    return ret
```

Inner functions

- Írjuk át úgy, hogy ne csinálja végig ha nem feltétlen szükséges!

Inner functions

- De hogy lehetne ezt?
- Mi is a baj?

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1  
    ret = num * fact_ver1(num - 1)  
    if ret >= 5000:  
        return 5000  
    return ret
```

Inner functions

- De hogy lehetne ezt?
- Mi is a baj?
- Az **elágaztatás** később van, mint a **rekurziós hívás**...

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1  
    ret = num * fact_ver1(num - 1)  
    if ret >= 5000:  
        return 5000  
    return ret
```

Inner functions

- De hogy lehetne ezt?
- Mi is a baj?
- Az **elágaztatás** később van, mint a **rekurziós hívás**...
- De nem tudjuk felcserélni a sorrendet, hisz a szorzat csak akkor derül ki ha a teljes hívási lánc lement!

```
def fact_ver1(num):  
    print("fact_ver1({})".format(num))  
    if num < 2:  
        return 1  
    ret = num * fact_ver1(num - 1)  
    if ret >= 5000:  
        return 5000  
    return ret
```

Inner functions

- Hmh... tároljuk a részeredményt!

Inner functions

- Ötlet: passzolgassuk a rész eredményt „accu” néven!

```
def fact_ver2(current, accu):
```

Inner functions

- Szokásos beköszönő print :)

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))
```

Inner functions

- Ha 0 vagy 1-nél vagyunk akkor ugye $0! = 1$ és $1! = 1$

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))
```

Inner functions

- Ha 0 vagy 1-nél vagyunk akkor ugye $0! = 1$ és $1! = 1$
- De nekünk mit is kéne csinálni 0 és 1 esetben?

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))
```

Inner functions

- Ha 0 vagy 1-nél vagyunk akkor ugye $0! = 1$ és $1! = 1$
- De nekünk mit is kéne csinálni 0 és 1 esetben?
- $\text{accu} * 0!$ vagy $\text{accu} * 1!$

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))
```

Inner functions

- Ha 0 vagy 1-nél vagyunk akkor ugye $0! = 1$ és $1! = 1$
- De nekünk mit is kéne csinálni 0 és 1 esetben?
- $\text{accu} * 0!$ vagy $\text{accu} * 1!$
- Dehát az $\text{accu} * 1$

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))
```

Inner functions

- Ha 0 vagy 1-nél vagyunk akkor ugye $0! = 1$ és $1! = 1$
- De nekünk mit is kéne csinálni 0 és 1 esetben?
- $\text{accu} * 0!$ vagy $\text{accu} * 1!$
- Dehát az $\text{accu} * 1$
- $1 * \text{something} = \text{something}$:)

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))
```

Inner functions

- Ha 0 vagy 1-nél vagyunk akkor ugye $0! = 1$ és $1! = 1$
- De nekünk mit is kéne csinálni 0 és 1 esetben?
- $\text{accu} * 0!$ vagy $\text{accu} * 1!$
- Dehát az $\text{accu} * 1$
- $1 * \text{something} = \text{something}$:)
- Adjuk vissza az accu -t mert az az eredmény!

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu
```


Inner functions

- Ha nem 0 és nem is 1 akkor mi a teendő?

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu
```

Inner functions

- Ha nem 0 és nem is 1 akkor mi a teendő?
- Az eddigi accu-t szorozni kell az épp mostani current-tel

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current
```

Inner functions

- Ha nem 0 és nem is 1 akkor mi a teendő?
- Az eddigi accu-t szorozni kell az épp mostani current-el
- Miért is? Nézzünk egy „futást”

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current
```

Inner functions

- Ha nem 0 és nem is 1 akkor mi a teendő?
- Az eddigi accu-t szorozni kell az épp mostani current-el
- Miért is? Nézzünk egy „futást”

4!

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current
```

Inner functions

- Ha nem 0 és nem is 1 akkor mi a teendő?
- Az eddigi accu-t szorozni kell az épp mostani current-el
- Miért is? Nézzünk egy „futást”

$$4! = 4 * 3!$$

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current
```

Inner functions

- Ha nem 0 és nem is 1 akkor mi a teendő?
- Az eddigi accu-t szorozni kell az épp mostani current-el
- Miért is? Nézzünk egy „futást”

$4! = 12 * 2!$

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current
```

Inner functions

- Ha nem 0 és nem is 1 akkor mi a teendő?
- Az eddigi accu-t szorozni kell az épp mostani current-el
- Miért is? Nézzünk egy „futást”

$$4! = 24 * 1!$$

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current
```

Inner functions

- Ha nem 0 és nem is 1 akkor mi a teendő?
- Az eddigi accu-t szorozni kell az épp mostani current-el
- Miért is? Nézzünk egy „futást”

4! = 24

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current
```


Inner functions

- És ugye tovább kell hívni „magunkat”, de az új részeredménnyel, illetve current-et eggyel csökkenteni

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
  
    return fact_ver2(current - 1, accu)
```

Inner functions

- Na de... mi lesz az egész 5000 =< dologgal?

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
  
    return fact_ver2(current - 1, accu)
```

Inner functions

- Mivel `accu` az eddigi részeredményt reprezentálja, ezért már tudjuk tesztelni!

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
  
    return fact_ver2(current - 1, accu)
```

Inner functions

- Mivel accu az eddigi részeredményt reprezentálja, ezért már tudjuk tesztelni!

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
    if accu >= 5000:  
        return 5000  
    return fact_ver2(current - 1, accu)
```

Inner functions

- Ok, teszteljük!

(link: https://github.com/rkeeves/munchausen/blob/main/src/fact_ver2.py)

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
    if accu >= 5000:  
        return 5000  
    return fact_ver2(current - 1, accu)
```

Inner functions

- Ok, teszteljük!

```
fact_ver2(8,1)
fact_ver2(7,8)
fact_ver2(6,56)
fact_ver2(5,336)
fact_ver2(4,1680)
```

```
def fact_ver2(current, accu):
    print("fact_ver2({}, {})".format(current, accu))
    if current < 2:
        return accu
    accu *= current
    if accu >= 5000:
        return 5000
    return fact_ver2(current - 1, accu)
```

Inner functions

- Ok, teszteljük!
- Sikerült levágni a hívás lánc egy részét!

```
fact_ver2(8,1)
fact_ver2(7,8)
fact_ver2(6,56)
fact_ver2(5,336)
fact_ver2(4,1680)
```

```
def fact_ver2(current, accu):
    print("fact_ver2({}, {})".format(current, accu))
    if current < 2:
        return accu
    accu *= current
    if accu >= 5000:
        return 5000
    return fact_ver2(current - 1, accu)
```

Inner functions

- De van egy bökkenő...hogya hívja user az új funkciót?

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
    if accu >= 5000:  
        return 5000  
    return fact_ver2(current - 1, accu)
```


Inner functions

- De van egy bökkenő...hogy hívja user az új funkciót?
- Honnan tudja a user, hogy neki az 1-et kell megadnia? (szorzás identitás)

```
def main():  
    fact_ver2(8, 1)  
    print("\nFinish")
```

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
    if accu >= 5000:  
        return 5000  
    return fact_ver2(current - 1, accu)
```

Inner functions

- Mi van ha 0-t ad meg?
- (tapasztalatom: ha lehet rossz inputot megadni az első dolog amit meg fognak adni az a rossz input lesz)

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
    if accu >= 5000:  
        return 5000  
    return fact_ver2(current - 1, accu)
```

Inner functions

- Mi van ha 0-t ad meg?
- (tapasztalatom: ha lehet rossz inputot megadni az első dolog amit meg fognak adni az a rossz input lesz)

```
fact_ver2(8,0)
fact_ver2(7,0)
fact_ver2(6,0)
fact_ver2(5,0)
fact_ver2(4,0)
fact_ver2(3,0)
fact_ver2(2,0)
fact_ver2(1,0)
```

```
def fact_ver2(current, accu):
    print("fact_ver2({}, {})".format(current, accu))
    if current < 2:
        return accu
    accu *= current
    if accu >= 5000:
        return 5000
    return fact_ver2(current - 1, accu)
```

Inner functions

- Mi van ha 0-t ad meg?
- (tapasztalatom: ha lehet rossz inputot megadni az első dolog amit meg fognak adni az a rossz input lesz)
- Oh boi...

```
fact_ver2(8,0)
fact_ver2(7,0)
fact_ver2(6,0)
fact_ver2(5,0)
fact_ver2(4,0)
fact_ver2(3,0)
fact_ver2(2,0)
fact_ver2(1,0)
```

```
def fact_ver2(current, accu):
    print("fact_ver2({}, {})".format(current, accu))
    if current < 2:
        return accu
    accu *= current
    if accu >= 5000:
        return 5000
    return fact_ver2(current - 1, accu)
```

Inner functions

- Eltudnánk-e érni, hogy a user NE TUDJON hülyeséget megadni?

Inner functions

- El tudnánk-e érni, hogy a user NE TUDJON hülyeséget megadni?
- El tudnánk-e zárni a külvilágtól a funkciót és így a user nem egyenesen ezt a funkciót hívna?

Inner functions

- Egymásba ágyazott függvények!

Inner functions

- Egymásba ágyazott függvények!

```
def fact_ver3(num):
```


Inner functions

- Egymásba ágyazott függvények!
- Fogjuk az előző függvényt

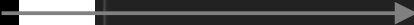
```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
    if accu >= 5000:  
        return 5000  
    return fact_ver2(current - 1, accu)
```

```
def fact_ver3(num):
```

Inner functions

- Egymásba ágyazott függvények!
- Fogjuk az előző függvényt
- És belepasszírozzuk a másikba

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
    if accu >= 5000:  
        return 5000  
    return fact_ver2(current - 1, accu)
```



```
def fact_ver3(num):
```

Inner functions

- Egymásba ágyazott függvények!
- Fogjuk az előző függvényt
- És belepasszírozzuk a másikba

```
def fact_ver2(current, accu):  
    print("fact_ver2({}, {})".format(current, accu))  
    if current < 2:  
        return accu  
    accu *= current  
    if accu >= 5000:  
        return 5000  
    return fact_ver2(current - 1, accu)
```

```
def fact_ver3(num):  
  
    def fact_ver2(current, accu):  
        print("fact_ver2({}, {})".format(current, accu))  
        if current < 2:  
            return accu  
        accu *→ current  
        if accu >= 5000:  
            return 5000  
        return fact_ver2(current - 1, accu)
```

Inner functions

- Ez egyébként sokkal tutibb dolgokra jó (closure-ök, higher order function-ök)
- De most ez nem kell

```
def fact_ver3(num):  
  
    def fact_ver2(current, accu):  
        print("fact_ver2({}, {})".format(current, accu))  
        if current < 2:  
            return accu  
        accu *= current  
        if accu >= 5000:  
            return 5000  
        return fact_ver2(current - 1, accu)
```

Inner functions

- És most már „kontrollált” körülmények között hívhatjuk!

```
def fact_ver3(num):  
  
    def fact_ver2(current, accu):  
        print("fact_ver2({}, {})".format(current, accu))  
        if current < 2:  
            return accu  
        accu *= current  
        if accu >= 5000:  
            return 5000  
        return fact_ver2(current - 1, accu)  
  
    return fact_ver2(num, 1)
```

Inner functions

- És most már „kontrollált” körülmények között hívhatjuk!
- User csak egy num-ot ad át

```
def fact_ver3(num):  
  
    def fact_ver2(current, accu):  
        print("fact_ver2({}, {})".format(current, accu))  
        if current < 2:  
            return accu  
        accu *= current  
        if accu >= 5000:  
            return 5000  
        return fact_ver2(current - 1, accu)  
  
    return fact_ver2(num, 1)
```

Inner functions

- És most már „kontrollált” körülmények között hívhatjuk!
- User csak egy num-ot ad át
- User nem tudja mi van „under the hood”

```
def fact_ver3(num):  
  
    def fact_ver2(current, accu):  
        print("fact_ver2({}, {})".format(current, accu))  
        if current < 2:  
            return accu  
        accu *= current  
        if accu >= 5000:  
            return 5000  
        return fact_ver2(current - 1, accu)  
  
    return fact_ver2(num, 1)
```

Inner functions

- Köszönést ne felejtjük el

```
def fact_ver3(num):  
    print("fact_ver3({})".format(num))  
  
    def fact_ver2(current, accu):  
        print("fact_ver2({}, {})".format(current, accu))  
        if current < 2:  
            return accu  
        accu *= current  
        if accu >= 5000:  
            return 5000  
        return fact_ver2(current - 1, accu)  
  
    return fact_ver2(num, 1)
```


Inner functions

- Teszteljük

(link: https://github.com/rkeeves/munchausen/blob/main/src/fact_ver3.py)

```
def fact_ver3(num):  
    print("fact_ver3({})".format(num))  
  
    def fact_ver2(current, accu):  
        print("fact_ver2({}, {})".format(current, accu))  
        if current < 2:  
            return accu  
        accu *= current  
        if accu >= 5000:  
            return 5000  
        return fact_ver2(current - 1, accu)  
  
    return fact_ver2(num, 1)
```

Inner functions

- Teszteljük

```
fact_ver3(8)
fact_ver2(8,1)
fact_ver2(7,8)
fact_ver2(6,56)
fact_ver2(5,336)
fact_ver2(4,1680)
```

```
def fact_ver3(num):
    print("fact_ver3({})".format(num))

    def fact_ver2(current, accu):
        print("fact_ver2({}, {})".format(current, accu))
        if current < 2:
            return accu
        accu *= current
        if accu >= 5000:
            return 5000
        return fact_ver2(current - 1, accu)

    return fact_ver2(num, 1)
```

Inner functions

- Teszteljük
- Működik!

```
fact_ver3(8)
fact_ver2(8,1)
fact_ver2(7,8)
fact_ver2(6,56)
fact_ver2(5,336)
fact_ver2(4,1680)
```

```
def fact_ver3(num):
    print("fact_ver3({})".format(num))

    def fact_ver2(current, accu):
        print("fact_ver2({}, {})".format(current, accu))
        if current < 2:
            return accu
        accu *= current
        if accu >= 5000:
            return 5000
        return fact_ver2(current - 1, accu)

    return fact_ver2(num, 1)
```

Inner functions

- Teszteljük
- Működik! Muaahahaha!

```
fact_ver3(8)
fact_ver2(8,1)
fact_ver2(7,8)
fact_ver2(6,56)
fact_ver2(5,336)
fact_ver2(4,1680)
```

```
def fact_ver3(num):
    print("fact_ver3({})".format(num))
```

```
def fact_ver2(current, accu):
    print("fact_ver2({},{})".format(current, accu))
```

```
    return fact_ver2(current - 1, accu)
```

```
    return fact_ver2(1, 1)
```



list_munchausens_of_decimal_places

- Most hogy ezen túl vagyunk térjünk vissza a count listák kilistázásra/generálásra/enumerálására

list_munchausens_of_decimal_places

- Nézzük meg akkor most az összes adott számjegyűt vizsgálót!

list_munchausens_of_decimal_places

- Nézzük meg akkor most az összes adott számjegyűt vizsgálót!
- Előre elnézést, kicsit hektikus lesz

list_munchausens_of_decimal_places

- Kap egy számot „decimal_places”

```
def list_munchausens_of_decimal_places(decimal_places):
```


list_munchausens_of_decimal_places

- Csináljunk egy belső függvényt (rekurzió)

```
def list_munchausens_of_decimal_places(decimal_places):
```

```
    def assign_count(digit_counts, remaining_count):
```

list_munchausens_of_decimal_places

- A belső függvény szépen rekurzívan fogja növelni a listát („digit_counts”)

```
def list_munchausens_of_decimal_places(decimal_places):
```

```
    def assign_count(digit_counts, remaining_count):
```

list_munchausens_of_decimal_places

- A „remaining_count” mondja meg mennyi darab számjegy még „kiosztható”

```
def list_munchausens_of_decimal_places(decimal_places):
```

```
    def assign_count(digit_counts, remaining_count):
```

list_munchausens_of_decimal_places

- Lényeg: Ez a dolog önmagát fogja hívni. Kell egy eset amikor megáll

```
def list_munchausens_of_decimal_places(decimal_places):
```

```
    def assign_count(digit_counts, remaining_count):
```

list_munchausens_of_decimal_places

- Mikor áll meg?
- Amikor „kész” a lista

```
def list_munchausens_of_decimal_places(decimal_places):
```

```
    def assign_count(digit_counts, remaining_count):
```

list_munchausens_of_decimal_places

- Mikor „kész” a lista?
- Mikor 10 eleme van

```
def list_munchausens_of_decimal_places(decimal_places):
```

```
    def assign_count(digit_counts, remaining_count):
```

list_munchausens_of_decimal_places

- Mikor „kész” a lista?
- Mikor 10 eleme van

```
def list_munchausens_of_decimal_places(decimal_places):  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:
```

list_munchausens_of_decimal_places

- Amikor „kész” mit csináljunk?

```
def list_munchausens_of_decimal_places(decimal_places):  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            return
```


list_munchausens_of_decimal_places

- Amikor „kész” mit csináljunk?
- Hát először is álljunk meg :)

```
def list_munchausens_of_decimal_places(decimal_places):  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
  
            return
```

list_munchausens_of_decimal_places

- De... mi az a pow_sum???

```
def list_munchausens_of_decimal_places(decimal_places):  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            if is_munchausen(digit_counts, pow_sum):  
                return
```

list_munchausens_of_decimal_places

- Még szerencse, hogy előre látóak voltunk és már megírtuk! :)

```
def list_munchausens_of_decimal_places(decimal_places):  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                return
```

list_munchausens_of_decimal_places

- Ha munchausen valami, akkor egy listába gyűjtjük...
- *Igen tudom, csúnya hogy outer scope-ból, de nem akartam belemenni a flat fold-ba*

```
def list_munchausens_of_decimal_places(decimal_places):  
  
    ...  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return
```

list_munchausens_of_decimal_places

- Ne feledjük el initelni a torzszülöttünket!

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return
```

list_munchausens_of_decimal_places

- Meg vagyunk a base case-el...
- Most jön az hogy mit teszünk akkor, ha még nincs kész a lista...

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return
```

list_munchausens_of_decimal_places

- Meg fogjuk magunkat hívni... duh
- De mivel is kell meghívni magunkat?

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Tegyük fel mi mondjuk a [0 1] listát kaptuk és még van remaining_count=2

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```


list_munchausens_of_decimal_places

- Tegyük fel mi mondjuk a [0 1] listát kaptuk és még van remaining_count=2
- Mi mondjuk úgy döntünk: 0 darabot választunk

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Tegyük fel mi mondjuk a [0 1] listát kaptuk és még van remaining_count=2
- Mi mondjuk úgy döntünk: 0 darabot választunk
- Ekkor [0 1 0] listával és remaining_count=2 vel kell hívni
- Hiszen nem használtunk fel egyet sem

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Tegyük fel mi mondjuk a [0 1] listát kaptuk és még van remaining_count=2
- De úgy is dönthetnénk mondjuk hogy: 1 darabot választunk

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Tegyük fel mi mondjuk a [0 1] listát kaptuk és még van remaining_count=2
- De úgy is dönthetnénk mondjuk hogy: 1 darabot választunk
- Ekkor [0 1 1] listával és remaining_count=1 vel kell hívni
- Hiszen felhasználtunk egyet

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Tegyük fel mi mondjuk a [0 1] listát kaptuk és még van remaining_count=2
- De lehetett volna 2 is...

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Olyan ismerős ez a 0, 1, 2 dolog... :)

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- For ciklus!

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        for chosen_count in range(1, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Azaz mondjuk kapunk `remaining_count = 2`-t
- Ekkor választhatunk ebből a számjegyből 0-t, 1-et, 2-t
- `remaining_count+1`-ig megyünk emiatt

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        for chosen_count in range(0, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```


list_munchausens_of_decimal_places

- Aprócska zökkenő: Ugye 0-tól 9-ig töltjük
- 9-nél választhatunk-e?

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        for chosen_count in range(0, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

`list_munchausens_of_decimal_places`

- Aprócska zökkenő: Ugye 0-tól 9-ig töltjük
- 9-nél választhatunk-e?
- Kicsit eltüntettem a kódot

`list_munchausens_of_decimal_places`

- Aprócska zökkenő: Ugye 0-tól 9-ig töltjük
- 9-nél választhatunk-e?
- Kicsit eltüntettem a kódot
- Tegyük fel 4 számjegy kell

list_munchausens_of_decimal_places

- Aprócska zökkenő: Ugye 0-tól 9-ig töltjük
- 9-nél választhatunk-e?
- Kicsit eltüntettem a kódot
- Tegyük fel 4 számjegy kell
- Mi már választottunk 2 db 1-est meg 1db 7-est mondjuk

list_munchausens_of_decimal_places

- Aprócska zökkenő: Ugye 0-tól 9-ig töltjük
- 9-nél választhatunk-e?
- Kicsit eltüntetem a kódot
- Tegyük fel 4 számjegy kell
- Mi már választottunk 2 db 1-est meg 1db 7-est mondjuk
- A többiből nyolcasig bezárólag nullát kértünk

`list_munchausens_of_decimal_places`

- Aprócska zökkenő: Ugye 0-tól 9-ig töltjük
- 9-nél választhatunk-e?
- Kicsit eltüntettem a kódot
- Tegyük fel 4 számjegy kell
- Mi már választottunk 2 db 1-est meg 1db 7-est mondjuk
- A többiből nyolcasig bezárólag nullát kértünk
- Ahhoz hogy meg legyen a 4 darab 9-esből kelleni fog 1

list_munchausens_of_decimal_places

- Aprócska zökkenő: Ugye 0-tól 9-ig töltjük
- 9-nél választhatunk-e?
- Kicsit eltüntetem a kódot
- Tegyük fel 4 számjegy kell
- Mi már választottunk 2 db 1-est meg 1db 7-est mondjuk
- A többiből nyolcasig bezárólag nullát kértünk
- Ahhoz hogy meg legyen a 4 darab 9-esből kelleni fog 1
- Azaz a kilencesnél nem választhatunk...

list_munchausens_of_decimal_places

- Vissza a kódhoz...

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        for chosen_count in range(1, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```


list_munchausens_of_decimal_places

- Mikor járunk a 9-es számjegy darabszám „eldöntésénél”?

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        for chosen_count in range(0, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Mikor járunk a 9-es számjegy darabszám „eldöntésénél”?
- Amikor a lista 9 elemű (Mert ugye 10 hosszú ha kész mínusz 1)

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        for chosen_count in range(0, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Ha kilencnél járunk mi történjen?

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
  
        for chosen_count in range(1, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Ha kilencnél járunk mi történjen?
- Rakja be a listába a maradékot...

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
        from_num = 0  
        if len(digit_counts) == 9:  
            from_num = remaining_count  
        for chosen_count in range(from_num, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Az alábbi hack ízlések és pofonok kérdése
- Lehetett volna explicit call-al is plusz return-nel is csinálni stb...

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
        from_num = 0  
        if len(digit_counts) == 9:  
            from_num = remaining_count  
        for chosen_count in range(from_num, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)
```

list_munchausens_of_decimal_places

- Visszaadjuk a listát, plusz én szeretek hülyeségre explicit terminálni

```
def list_munchausens_of_decimal_places(decimal_places):
    ret = []

    if decimal_places < 1:
        return ret

    def assign_count(digit_counts, remaining_count):
        if len(digit_counts) == 10:
            pow_sum = compute_pow_sum(digit_counts)
            if is_munchausen(digit_counts, pow_sum):
                ret.append(pow_sum)
            return
        from_num = 0
        if len(digit_counts) == 9:
            from_num = remaining_count
        for chosen_count in range(from_num, remaining_count + 1):
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)

    return ret
```

list_munchausens_of_decimal_places

- Szuper
- De hogy indítsuk be?

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    if decimal_places < 1:  
        return ret  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
        from_num = 0  
        if len(digit_counts) == 9:  
            from_num = remaining_count  
        for chosen_count in range(from_num, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)  
  
    return ret
```

list_munchausens_of_decimal_places

- Milyen listát és milyen remaining_count-ot kapjon először?

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    if decimal_places < 1:  
        return ret  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
        from_num = 0  
        if len(digit_counts) == 9:  
            from_num = remaining_count  
        for chosen_count in range(from_num, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)  
  
    return ret
```


list_munchausens_of_decimal_places

- Milyen listát és milyen remaining_count-ot kapjon először?
- Üres lista és pont annyi a remaining_count amennyi decimal_places hely van!

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    if decimal_places < 1:  
        return ret  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
        from_num = 0  
        if len(digit_counts) == 9:  
            from_num = remaining_count  
        for chosen_count in range(from_num, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)  
  
    assign_count([], decimal_places)  
    return ret
```

list_munchausens_of_decimal_places

- Ok, fele kész
- Még kell az ezt hívó...

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = []  
  
    if decimal_places < 1:  
        return ret  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            pow_sum = compute_pow_sum(digit_counts)  
            if is_munchausen(digit_counts, pow_sum):  
                ret.append(pow_sum)  
            return  
        from_num = 0  
        if len(digit_counts) == 9:  
            from_num = remaining_count  
        for chosen_count in range(from_num, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)  
  
    assign_count([], decimal_places)  
    return ret
```

list_munchausens_up_to_decimal_places

- Legyen a neve a költőí „list_munchausens_up_to_decimal_places”

```
def list_munchausens_up_to_decimal_places(max_decimal_places):
```

list_munchausens_up_to_decimal_places

- Megkapjuk max hány számjegyesig „max_decimal_places”

```
def list_munchausens_up_to_decimal_places(max_decimal_places):
```

list_munchausens_up_to_decimal_places

- Végig iterálunk a decimal_place-eken (0-t ugye kihagyjuk :)

```
def list_munchausens_up_to_decimal_places(max_decimal_places):  
    for decimal_places in range(1, max_decimal_places + 1):
```

list_munchausens_up_to_decimal_places

- Most nem append lesz
- Miért nem?
- Mert listát ad vissza a hívott függvény
- Szóval append helyett concat-oljuk az accu listához

```
def list_munchausens_up_to_decimal_places(max_decimal_places):  
    for decimal_places in range(1, max_decimal_places + 1):  
        accu += list_munchausens_of_decimal_places(decimal_places)
```

list_munchausens_up_to_decimal_places

- Initeljük, és adjuk vissza a listát
- Meg egy sort-ot is, hogy azt is használjuk egyszer :)

```
def list_munchausens_up_to_decimal_places(max_decimal_places):  
    accu = []  
    for decimal_places in range(1, max_decimal_places + 1):  
        accu += list_munchausens_of_decimal_places(decimal_places)  
    return sorted(accu)
```

list_munchausens_up_to_decimal_places

- Jó, most már akkor futassuk is, hisz ez a végleges progí!

```
def list_munchausens_up_to_decimal_places(max_decimal_places):  
    accu = []  
    for decimal_places in range(1, max_decimal_places + 1):  
        accu += list_munchausens_of_decimal_places(decimal_places)  
    return sorted(accu)
```


list_munchausens_up_to_decimal_places

- Teszt

(link: https://github.com/rkeeves/munchausen/blob/main/src/e08_munchausen_property.py)

list_munchausens_up_to_decimal_places

- Teszt
- Ok-nak néz ki

```
0.000000s Decimals places: 1 -> [0, 1]
0.000000s Decimals places: 2 -> [0, 1]
0.001965s Decimals places: 3 -> [0, 1]
0.004984s Decimals places: 4 -> [0, 1, 3435]
0.010010s Decimals places: 5 -> [0, 1, 3435]
0.030915s Decimals places: 6 -> [0, 1, 3435]
0.074804s Decimals places: 7 -> [0, 1, 3435]
0.167554s Decimals places: 8 -> [0, 1, 3435]
0.345081s Decimals places: 9 -> [0, 1, 3435, 438579088]
0.682585s Decimals places: 10 -> [0, 1, 3435, 438579088]
1.259651s Decimals places: 11 -> [0, 1, 3435, 438579088]
```

list_munchausens_up_to_decimal_places

- Teszt
- Várjunk... MILYEN EXEC TIME?

```
0.000000s Decimals places: 1 -> [0, 1]
0.000000s Decimals places: 2 -> [0, 1]
0.001965s Decimals places: 3 -> [0, 1]
0.004984s Decimals places: 4 -> [0, 1, 3435]
0.010010s Decimals places: 5 -> [0, 1, 3435]
0.030915s Decimals places: 6 -> [0, 1, 3435]
0.074804s Decimals places: 7 -> [0, 1, 3435]
0.167554s Decimals places: 8 -> [0, 1, 3435]
0.345081s Decimals places: 9 -> [0, 1, 3435, 438579088]
0.682585s Decimals places: 10 -> [0, 1, 3435, 438579088]
1.259651s Decimals places: 11 -> [0, 1, 3435, 438579088]
```

list_munchausens_up_to_decimal_places

- Teszt
- Várjunk... MILYEN EXEC TIME?

```
0.000000s Decimals places: 1 -> [0, 1]
0.000000s Decimals places: 2 -> [0, 1]
0.001965s Decimals places: 3 -> [0, 1]
0.004984s Decimals places: 4 -> [0, 1, 3435]
0.010010s Decimals places: 5 -> [0, 1, 3435]
0.030915s Decimals places: 6 -> [0, 1, 3435]
0.074804s Decimals places: 7 -> [0, 1, 3435]
0.167554s Decimals places: 8 -> [0, 1, 3435]
0.345081s Decimals places: 9 -> [0, 1, 3435, 438579088]
0.682585s Decimals places: 10 -> [0, 1, 3435, 438579088]
1.259651s Decimals places: 11 -> [0, 1, 3435, 438579088]
```

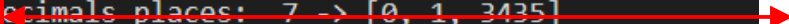
[29.029986s] 10000000 -> [0, 1, 3435]

0.07s vs. 29.03s

list_munchausens_up_to_decimal_places

- Teszt
- Várjunk... MILYEN EXEC TIME?

```
0.000000s Decimals places: 1 -> [0, 1]
0.000000s Decimals places: 2 -> [0, 1]
0.001965s Decimals places: 3 -> [0, 1]
0.004984s Decimals places: 4 -> [0, 1, 3435]
0.010010s Decimals places: 5 -> [0, 1, 3435]
0.030915s Decimals places: 6 -> [0, 1, 3435]
0.074804s Decimals places: 7 -> [0, 1, 3435]
0.167554s Decimals places: 8 -> [0, 1, 3435]
0.345081s Decimals places: 9 -> [0, 1, 3435, 438579088]
0.682585s Decimals places: 10 -> [0, 1, 3435, 438579088]
1.259651s Decimals places: 11 -> [0, 1, 3435, 438579088]
```



```
[29.029986s] 10000000 -> [0, 1, 3435]
```

0.07s vs. 29.03s (mindezt úgy hogy str-rel számoljuk ovis módra a számjegyeket muahahahaha)

`list_munchausens_up_to_decimal_places`

- A lényeg, hogy a trükk legyen érthető
- Szénné lehet optimalni még ezen túl is

További optimalizálás

- `str(num)...` miért nem %?

További optimalizálás

- `str(num)...` miért nem %?
- Biztos végig kell menni a teljes leszámláláson?
Lehetne esetleg gyorsabban terminálni a leszámlálási loop-ban False-szal?

További optimalizálás

- `str(num)...` miért nem %?
- Biztos végig kell menni a teljes leszámláláson?
Lehetne esetleg gyorsabban terminálni a leszámlálási loopban False-al?
- Nem lehetne a hatvány rész szummákat számolni és így nem kéne a call tree leaf-jein hívogatni azt a buta `compute_pow_sum`-ot?

Összefoglaló

- Szóval lehetne még optimalizálni, de...

Összefoglaló

- Szóval lehetne még optimalizálni, de sok volt ez így egyszerre...



Köszönöm a figyelmet és a kitartást!

Mazochistáknak

- Van-e valami IGAZI limit?
- Elképzelhető-e, hogy egy idő után nem tud elég gyorsan nőni a szumma?
- Ha kilenceseket használunk úgy kapjuk a legnagyobb szummát
- $9^9 = 387\,420\,489$

Mazochistáknak

- A szumma a számjegyek növelésével lineárisan nő

	Szumma	Számjegy

Mazochistáknak

- A szumma a számjegyek növelésével lineárisan nő

	Szumma	Számjegy
1	$1 * 9^9 = 387\,420\,489$	9

--	--	--

--	--	--

Mazochistáknak

- A szumma a számjegyek növelésével lineárisan nő

	Szumma	Számjegy
1	$1 * 9^9 = 387\,420\,489$	9
2	$2 * 9^9 = 774\,840\,978$	99

--	--	--

Mazochistáknak

- A szumma a számjegyek növelésével lineárisan nő

	Szumma	Számjegy
1	$1 * 9^9 = 387\,420\,489$	9
2	$2 * 9^9 = 774\,840\,978$	99
...

--	--	--

Mazochistáknak

- A szumma a számjegyek növelésével lineárisan nő

	Szumma	Számjegy
1	$1 * 9^9 = 387\,420\,489$	9
2	$2 * 9^9 = 774\,840\,978$	99
...
10	$10 * 9^9 = 3\,874\,204\,890$	9 999 999 999

Mazochistáknak

- A szumma a számjegyek növelésével lineárisan nő

	Szumma	Számjegy
1	$1 * 9^9 = 387\,420\,489$	9
2	$2 * 9^9 = 774\,840\,978$	99
...
10	$10 * 9^9 = 3\,874\,204\,890$	9 999 999 999
11	$11 * 9^9 = 4\,261\,625\,379$	99 999 999 999

Mazochistáknak

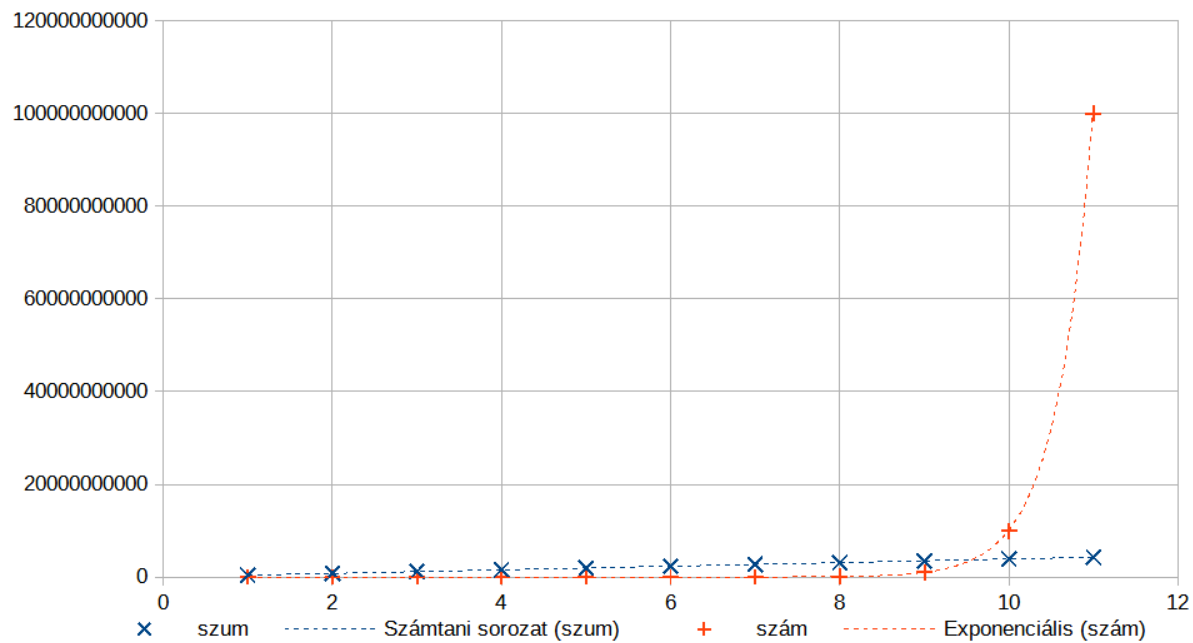
- A szumma a számjegyek növelésével lineárisan nő

	Szumma	Számjegy
1	$1 * 9^9 = 387\,420\,489$	9
2	$2 * 9^9 = 774\,840\,978$	99
...
10	$10 * 9^9 = 3\,874\,204\,890$	9 999 999 999
11	$11 * 9^9 = 4\,261\,625\,379$	99 999 999 999

Egy „nagyságrenddel” kisebb!

Mazochistáknak

- Grafikusabban



Mazochistáknak

- De nem csak felső korlátra jó
- Például 9-es $9^9 = 387\,420\,489$

Mazochistáknak

- De nem csak felső korlátra jó
- Például 9-es $9^9 = 387\,420\,489$
- Azaz 9-est $387\,420\,489$ alatt időkidobás felhasználni

Mazochistáknak

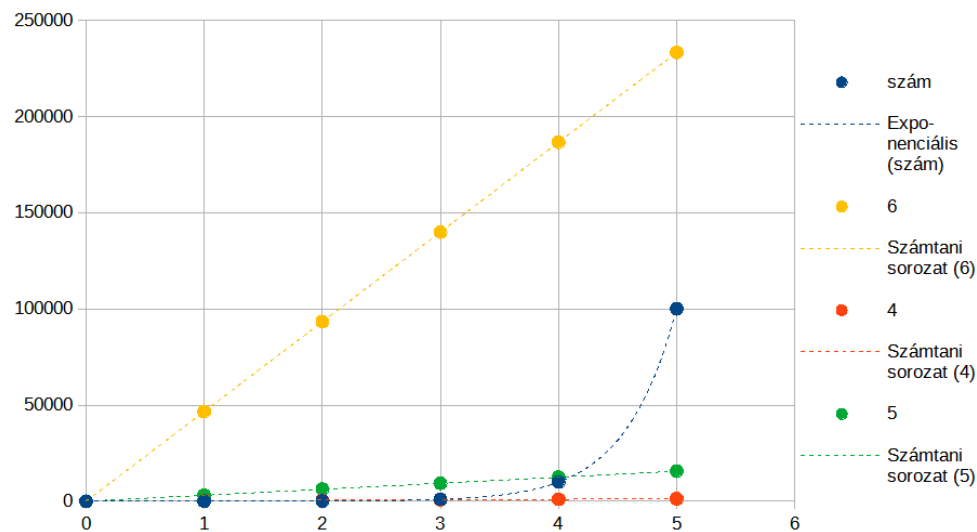
- De nem csak felső korlátra jó
- Például 9-es $9^9 = 387\,420\,489$
- Azaz 9-est $387\,420\,489$ alatt időkidobás felhasználni
- Azaz 8-ast $16\,777\,216$ alatt időkidobás felhasználni

Mazochistáknak

- De nem csak felső korlátra jó
- Például 9-es $9^9 = 387\,420\,489$
- Azaz 9-est $387\,420\,489$ alatt időkidobás felhasználni
- Azaz 8-ast $16\,777\,216$ alatt időkidobás felhasználni
- Stb.
- Szóval bizonyos szám alatt ha valamiben mondjuk van egy kilences akkor „átugorhatnánk”
- Még kevesebb számolás!

Mazochistáknak

- De nem csak felső korlátra jó
- Például 9-es $9^9 = 387\,420\,489$
- Azaz 9-est 387 420 489 alatt időkidobás felh
- Azaz 8-ast 16 777 216 alatt időkidobás felh
- Stb.
- Szóval bizonyos szám alatt ha valamiben n
- van egy kilences akkor „átugorhatnánk”
- Még kevesebb számolás!



Mazochistáknak

- Az algoritmus bugos!

Mazochistáknak

- Az algoritmus bugos!
- Számítsuk ki hányszor kéne lefutnia 3 számjegyre

Mazochistáknak

- Az algoritmus bugos!
- Számítsuk ki hányszor kéne lefutnia 3 számjegyre

$$\left(\binom{10+1-1}{1}\right) = 10$$

$$\left(\binom{10+2-1}{2} - 1\right) = 55 - 1 = 54$$

$$\left(\binom{10+3-1}{3} - 1\right) = 220 - 1 = 219$$

$$10 + 54 + 219 = 283$$

Mazochistáknak

- Az algoritmus bugos!
- Számítsuk ki hányszor kéne lefutnia 3 számjegyre

$$\left(\binom{10+1-1}{1}\right) = 10$$

$$\left(\binom{10+2-1}{2} - 1\right) = 55 - 1 = 54$$

$$\left(\binom{10+3-1}{3} - 1\right) = 220 - 1 = 219$$

$$10 + 54 + 219 = 283$$

- Teszteljük

(link: https://github.com/rkeeves/munchausen/blob/main/src/e09_munchausen_property.py)

Mazochistáknak

- Az algoritmus bugos!
- Számítsuk ki hányszor kéne lefutnia 3 számjegyre

$$\binom{10+1-1}{1} = 10$$

$$\binom{10+2-1}{2} - 1 = 55 - 1 = 54$$

$$\binom{10+3-1}{3} - 1 = 220 - 1 = 219$$

$$10 + 54 + 219 = 283$$

- Teszteljük

```
max_decimal_places -> total_case_count
1 -> 10
2 -> 65
3 -> 285
4 -> 1000
5 -> 3002
6 -> 8007
7 -> 19447
8 -> 43757
9 -> 92377
10 -> 184755
11 -> 352715
Finished
```

Mazochistáknak

- Az algoritmus bugos!
- Számítsuk ki hányszor kéne lefutnia 3 számjegyre

$$\binom{10+1-1}{1} = 10$$

$$\binom{10+2-1}{2} - 1 = 55 - 1 = 54$$

$$\binom{10+3-1}{3} - 1 = 220 - 1 = 219$$

$$10 + 54 + 219 = 283$$

```
max_decimal_places -> total_case_count
1 -> 10
2 -> 65
3 -> 285
4 -> 1000
5 -> 3002
6 -> 8007
7 -> 19447
8 -> 43757
9 -> 92377
10 -> 184755
11 -> 352715
Finished
```

- Teszteljük
- Hát itt valami nem jó...

Mazochistáknak

- Az algoritmus bugos!
- Számítsuk ki hányszor kéne lefutnia 3 számjegyre

$$\binom{10+1-1}{1} = 10$$

$$\binom{10+2-1}{2} - 1 = 55 - 1 = 54$$

$$\binom{10+3-1}{3} - 1 = 220 - 1 = 219$$

$$10 + 54 + 219 = 283$$

```
max_decimal_places -> total_case_count
1 -> 10
2 -> 65
3 -> 285
4 -> 1000
5 -> 3002
6 -> 8007
7 -> 19447
8 -> 43757
9 -> 92377
10 -> 184755
11 -> 352715
Finished
```

- Teszteljük
- Mi lehet az oka?

Mazochistáknak

- Az algoritmus bugos!
- Számítsuk ki hányszor kéne lefutnia 3 számjegyre

$$\left(\binom{10+1-1}{1} - 1 \right) = 9$$

$$\left(\binom{10+2-1}{2} - 1 \right) = 55 - 1 = 54$$

$$\left(\binom{10+3-1}{3} - 1 \right) = 220 - 1 = 219$$

$$(9 + 54 + 219 + 1) = 283$$

```
max_decimal_places -> total_case_count
1 -> 10
2 -> 65
3 -> 285
4 -> 1000
5 -> 3002
6 -> 8007
7 -> 19447
8 -> 43757
9 -> 92377
10 -> 184755
11 -> 352715
Finished
```

- Teszteljük
- Át tudjuk-e írni, hogy ne csinálja?

Mazochistáknak

- Lehetne általánosabbra!

Mazochistáknak

- Lehetne általánosabbra!
- Ugye **ő** hívja **öt**, és átadja hány számjegy lehessen

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = [0]  
  
    if decimal_places < 1:  
        return ret  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            ret[0] += 1  
            return  
        from_num = 0  
        if len(digit_counts) == 9:  
            from_num = remaining_count  
        for chosen_count in range(from_num, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)  
  
    assign_count([], decimal_places)  
    return ret[0]  
  
def list_munchausens_up_to_decimal_places(max_decimal_places):  
    total_case_count = 0  
    for decimal_places in range(1, max_decimal_places + 1):  
        total_case_count += list_munchausens_of_decimal_places(decimal_places)  
    print("{:2} -> {}".format(max_decimal_places, total_case_count))  
    return total_case_count
```

Mazochistáknak

- Lehetne általánosabbra!
- Ugye **ő** hívja **öt**, és átadja hány számjegy lehessen
- Mi lenne, ha beletudnánk kódolni a pirosat a zöldbe?

```
def list_munchausens_of_decimal_places(decimal_places):  
    ret = [0]  
  
    if decimal_places < 1:  
        return ret  
  
    def assign_count(digit_counts, remaining_count):  
        if len(digit_counts) == 10:  
            ret[0] += 1  
            return  
        from_num = 0  
        if len(digit_counts) == 9:  
            from_num = remaining_count  
        for chosen_count in range(from_num, remaining_count + 1):  
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)  
  
    assign_count([], decimal_places)  
    return ret[0]  
  
def list_munchausens_up_to_decimal_places(max_decimal_places):  
    total_case_count = 0  
    for decimal_places in range(1, max_decimal_places + 1):  
        total_case_count += list_munchausens_of_decimal_places(decimal_places)  
    print("{:2} -> {}".format(max_decimal_places, total_case_count))  
    return total_case_count
```

Mazochistáknak

- Lehetne általánosabbra!
- Ugye **ő** hívja **öt**, és átadja hány számjegy lehessen
- Mi lenne, ha beletudnánk kódolni a pirosat a zöldbe?
- Tipp:
Ismerjük-e a kifejezést „leading zero”?

```
def list_munchausens_of_decimal_places(decimal_places):
    ret = [0]

    if decimal_places < 1:
        return ret

    def assign_count(digit_counts, remaining_count):
        if len(digit_counts) == 10:
            ret[0] += 1
            return
        from_num = 0
        if len(digit_counts) == 9:
            from_num = remaining_count
        for chosen_count in range(from_num, remaining_count + 1):
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)

    assign_count([], decimal_places)
    return ret[0]

def list_munchausens_up_to_decimal_places(max_decimal_places):
    total_case_count = 0
    for decimal_places in range(1, max_decimal_places + 1):
        total_case_count += list_munchausens_of_decimal_places(decimal_places)
    print("{:2} -> {}".format(max_decimal_places, total_case_count))
    return total_case_count
```

Mazochistáknak

- Lehetne általánosabbra!
- Ugye **ő** hívja **őt**, és átadja hány számjegy lehessen
- Mi lenne, ha beletudnánk kódolni a pirosat a zöldbe?
- Tipp:
Ismerjük-e a kifejezést „leading zero”?
- Senki nem mondta, hogy csak 10 elemű lehet a vektor(csak így „more straightforward”-abb volt)

```
def list_munchausens_of_decimal_places(decimal_places):
    ret = [0]

    if decimal_places < 1:
        return ret

    def assign_count(digit_counts, remaining_count):
        if len(digit_counts) == 10:
            ret[0] += 1
            return
        from_num = 0
        if len(digit_counts) == 9:
            from_num = remaining_count
        for chosen_count in range(from_num, remaining_count + 1):
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)

    assign_count([], decimal_places)
    return ret[0]

def list_munchausens_up_to_decimal_places(max_decimal_places):
    total_case_count = 0
    for decimal_places in range(1, max_decimal_places + 1):
        total_case_count += list_munchausens_of_decimal_places(decimal_places)
    print("{:2} -> {}".format(max_decimal_places, total_case_count))
    return total_case_count
```

Mazochistáknak

- A rekurzió „vissza írható” normál loop-á
- Lehet is előnye!
- (Mármint csak gondoljunk arra, hogy egy csomó könyvelés a hívogatásokat adminisztrálni)

```
def list_munchausens_of_decimal_places(decimal_places):
    ret = [0]

    if decimal_places < 1:
        return ret

    def assign_count(digit_counts, remaining_count):
        if len(digit_counts) == 10:
            ret[0] += 1
            return
        from_num = 0
        if len(digit_counts) == 9:
            from_num = remaining_count
        for chosen_count in range(from_num, remaining_count + 1):
            assign_count(digit_counts + [chosen_count], remaining_count - chosen_count)

    assign_count([], decimal_places)
    return ret[0]

def list_munchausens_up_to_decimal_places(max_decimal_places):
    total_case_count = 0
    for decimal_places in range(1, max_decimal_places + 1):
        total_case_count += list_munchausens_of_decimal_places(decimal_places)
    print("{:2} -> {}".format(max_decimal_places, total_case_count))
    return total_case_count
```